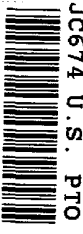


12/29/99



Please type a plus sign (+) inside this box → ☒

Approved for use through 09/30/2000 OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number

UTILITY PATENT APPLICATION TRANSMITTAL (Only for new nonprovisional applications under 37 C.F.R. § 1.53(b))	Attorney Docket No. 37081X00 219.3 D81K00
	First Inventor or Application Identifier Amit A. Merchant
	Title See 1 in Addendum
	Express Mail Label No.

APPLICATION ELEMENTS <i>See MPEP chapter 600 concerning utility patent application contents</i>	ADDRESS TO: Assistant Commissioner for Patents Box Patent Application Washington, DC 20231
1. <input checked="" type="checkbox"/> * Fee Transmittal Form (e.g., PTO/SB/17) (Submit an original and a duplicate for fee processing)	5. <input type="checkbox"/> Microfiche Computer Program (Appendix)
2. <input checked="" type="checkbox"/> Specification [Total Pages 30] (preferred arrangement set forth below) <ul style="list-style-type: none">- Descriptive title of the invention- Cross References to Related Applications- Statement Regarding Fed sponsored R & D- Reference to Microfiche Appendix- Background of the invention- Brief Summary of the invention- Brief Description of the Drawings (if filed)- Detailed Description- Claim(s)- Abstract of the Disclosure	6. Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary) <ul style="list-style-type: none">a. <input type="checkbox"/> Computer Readable Copyb. <input type="checkbox"/> Paper Copy (identical to computer copy)c. <input type="checkbox"/> Statement verifying identity of above copies
3. <input checked="" type="checkbox"/> Drawing(s) (35 U.S.C. 113) [Total Sheets 3]	ACCOMPANYING APPLICATION PARTS 7. <input checked="" type="checkbox"/> Assignment Papers (cover sheet & document(s)) 8. <input type="checkbox"/> 37 C.F.R. § 3.73(b) Statement (when there is an assignee) <input checked="" type="checkbox"/> Power of Attorney 9. <input type="checkbox"/> English Translation Document (if applicable) 10. <input type="checkbox"/> Information Disclosure Statement (IDS)/PTO-1449 <input type="checkbox"/> Copies of IDS Citations 11. <input type="checkbox"/> Preliminary Amendment 12. <input checked="" type="checkbox"/> Return Receipt Postcard (MPEP 503) (Should be specifically itemized) 13. <input type="checkbox"/> * Small Entity Statement(s) <input type="checkbox"/> Statement filed in prior application (PTO/SB/09-12) Status still proper and desired 14. <input type="checkbox"/> Certified Copy of Priority Document(s) (if foreign priority is claimed) 15. <input type="checkbox"/> Other:
4. Oath or Declaration [Total Pages 5] <ul style="list-style-type: none">a. <input checked="" type="checkbox"/> Newly executed (original or copy)b. <input type="checkbox"/> Copy from a prior application (37 C.F.R. § 1.63(d)) (for continuation/divisional with Box 16 completed)<ul style="list-style-type: none">i. <input type="checkbox"/> DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b)	
* NOTE FOR ITEMS 1 & 13 IN ORDER TO BE ENTITLED TO PAY SMALL ENTITY FEES, A SMALL ENTITY STATEMENT IS REQUIRED (37 C.F.R. § 1.27), EXCEPT IF ONE FILED IN A PRIOR APPLICATION IS RELIED UPON (37 C.F.R. § 1.28).	

16. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in a preliminary amendment.
☐ Continuation ☐ Divisional ☒ Continuation-in-part (CIP) of prior application No. **09 / 106,857**
Prior application information: Examiner _____ Group / Art Unit _____
For CONTINUATION or DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 4b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

17. CORRESPONDENCE ADDRESS				
<input checked="" type="checkbox"/> Customer Number or Bar Code Label		020457		or <input type="checkbox"/> Correspondence address below
(Insert Customer No. or Attach bar code label here)				
Name				
Address				
City	State	Zip Code		
Country	Telephone	Fax		

Name (Print/Type)	R. Edward Brake	Registration No. (Attorney/Agent)	37,784
Signature	<i>R. Edward Brake</i>	Date	Dec. 29, 1999

Burden Hour Statement This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231

[illegible]

- [illegible]

219.37081X00
P5600

UNITED STATES PATENT APPLICATION
FOR

**A PROCESSOR WITH A REPLAY SYSTEM THAT INCLUDES A REPLAY QUEUE
FOR IMPROVED THROUGHPUT**

INVENTORS:

AMIT A. MERCHANT

DARRELL D. BOGGS

DAVID J. SAGER

PREPARED BY:

ANTONELLI, TERRY, STOUT & KRAUS, LLP

SUITE 1800

1300 NORTH SEVENTEENTH STREET

ARLINGTON, VA 22209

(703) 312-6600

FAX (703) 312-6666

**A PROCESSOR WITH A REPLAY SYSTEM THAT INCLUDES A REPLAY QUEUE
FOR IMPROVED THROUGHPUT**

Cross-Reference To Related Applications

This application is a continuation-in-part of U.S. patent application serial no. 09/106,857, filed June 30, 1998 and entitled "Computer Processor With a Replay System" which is a continuation-in-part of application serial no. 08/746,547 filed November 13, 1996 entitled "Processor Having Replay Architecture" now U.S. Patent No. 5,966,544.

Field

The invention generally relates to processors, and in particular to a replay system having a replay queue.

Background

The primary function of most computer processors is to execute a stream of computer instructions that are retrieved from a storage device. Many processors are designed to fetch an instruction and execute that instruction before fetching the next instruction. Therefore, with these processors, there is an assurance that any register or memory value that is modified or retrieved by

a given instruction will be available to instructions following it. For example, consider the following set of instructions:

- 1) Load memory-1 \rightarrow register-X;
- 2) Add1 register-X register-Y \rightarrow register-Z;
- 5 3) Add2 register-Y register-Z \rightarrow register-W.

The first instruction loads the content of memory-1 into register-X. The second instruction adds the content of register-X to the content of register-Y and stores the result in register-Z. The third instruction adds the content of register-Y to the content of register-Z and stores the result in register-W. In this set of instructions, instructions 2 and 3 are considered "dependent" instructions that are dependent on instruction 1. In other words, if register-X is not loaded with valid data in instruction 1 before instructions 2 and 3 are executed, instructions 2 and 3 will generate improper results.

With the traditional "fetch and execute" processors, the second instruction will not be executed until the first instruction has properly executed. For example, the second instruction may not be dispatched to the processor until a cache hit/miss signal is received as a result of the first instruction.

Further, the third instruction will not be dispatched until an indication that the second instruction has properly executed has been received. Therefore, it can be seen that this short program cannot be executed in less time than $T=L_1 + L_2 + L_3$, where L_1 , L_2 and L_3 represent the latency of the three instructions. Hence, to ultimately execute the program faster, it will be necessary to reduce the latencies of the instructions.

Therefore, there is a need for a computer processor that can schedule and execute instructions with improved speed to reduce latencies.

Summary

According to an embodiment of the present invention, a processor is provided that includes an execution unit to execute instructions and a replay system coupled to the execution unit to replay instructions which have not executed properly. The replay system includes a checker to determine whether each instruction has executed properly and a replay queue coupled to the checker to temporarily store one or more instructions for replay.

Brief Description of the Drawings

The foregoing and a better understanding of the present invention will become apparent from the following detailed description of exemplary embodiments and the claims when read in connection with the accompanying drawings, all forming a part of the disclosure of this invention. While the foregoing and following written and illustrated disclosure focuses on disclosing example embodiments of the invention, it should be clearly understood that the same is by way of illustration and example only and is not limited thereto. The spirit and scope of the present invention being limited only by the terms of the appended claims.

The following represents brief descriptions of the drawings, wherein:

Fig. 1 is a block diagram illustrating a computer system that includes a processor according to an embodiment of the present invention.

Fig. 2 is a flow chart illustrating an example operation of instruction processing.

Fig. 3 is a diagram illustrating an example format of an instruction provided in a replay path according to an embodiment of the present invention.

Detailed Description

I. Introduction

According to an embodiment of the present invention, a processor is provided that speculatively schedules instructions for execution and includes a replay system. Speculative scheduling allows the scheduling latency for instructions to be reduced. The replay system replays instructions that were not correctly executed when they were originally dispatched to an execution unit. For example, a memory load instruction may not execute properly if there is a L0 cache miss during execution, thereby requiring the instruction to be replayed (or re-executed).

However, one challenging aspect of such a replay system is the possibility for long latency instructions to circulate through the replay system and re-execute many times before executing properly. One example of a long latency instruction could be a memory load instruction in which there is a L0 cache miss and a L1 cache miss (i.e., on-chip cache miss) on the first execution attempt. As a result, the execution unit may then retrieve the data from an external memory device across an external bus which can be very time consuming (e.g., requiring several hundred clock cycles). The unnecessary and repeated re-execution of this long latency load instruction before its source data has returned wastes valuable execution resources, prevents other instructions from executing and increases application latency.

Therefore, according to an embodiment, a replay queue is provided for temporarily storing the long latency instruction and its dependent instructions. When the long latency instruction is ready for execution (e.g., when the source data for a memory load instruction returns from external

memory), the long latency instruction and the dependent instructions can then be unloaded from the replay queue for execution.

II. Overall System Architecture

Fig. 1 is a block diagram illustrating a computer system that includes a processor according to an embodiment. The processor 100 includes a Front End 112, which may include several units, such as an instruction fetch unit, an instruction decoder for decoding instructions (e.g., for decoding complex instructions into one or more micro-operations or uops), a Register Alias Table (RAT) for mapping logical registers to physical registers for source operands and the destination, and an instruction queue (IQ) for temporarily storing instructions. In one embodiment, the instructions stored in the instruction queue are micro-operations or uops, but other types of instructions can be used. The Front End 112 may include different or even additional units. According to an embodiment, each instruction includes up to two logical sources and one logical destination. The sources and destination are logical registers within the processor 100. The RAT within the Front End 112 may map logical sources and destinations to physical sources and destinations, respectively.

Front End 112 is coupled to a scheduler 114. Scheduler 114 dispatches instructions received from the processor Front End 112 (e.g., from the instruction queue of the Front End 112) when the resources are available to execute the instructions. Normally, scheduler 114 sends out a continuous stream of instructions. However, scheduler 114 is able to detect, by itself or by receiving a signal, when an instruction should not be dispatched. When scheduler 114 detects this, it does not dispatch an instruction in the next clock cycle. When an instruction is not dispatched, a "hole" is formed in the instruction stream from the scheduler 114, and another device can insert an instruction in the

hole. The instructions are dispatched from scheduler 114 speculatively. Therefore, scheduler 114 can dispatch an instruction without first determining whether data needed by the instruction is valid or available.

Scheduler 114 outputs the instructions to a dispatch multiplexer (mux) 116. The output of mux 116 includes two parallel paths, including an execution path (beginning at line 137) and a replay path (beginning at line 139). The execution path will be briefly described first, while the replay path will be described below in connection with a description of a replay system 117.

The output of the multiplexer 116 is coupled to an execution unit 118. Execution unit 118 executes received instructions. Execution unit 118 can be an arithmetic logic unit ("ALU"), a floating point ALU, a memory unit for performing memory loads (memory data reads) and stores (memory data writes), etc. In the embodiment shown in Fig. 1, execution unit 118 is a memory load unit that is responsible for loading data stored in a memory device to a register (i.e., a data read from memory).

Execution unit 118 is coupled to multiple levels of memory devices that store data. First, execution unit 118 is directly coupled to an L0 cache system 120, which may also be referred to as a data cache. As described herein, the term "cache system" includes all cache related components, including cache memory, and cache TAG memory and hit/miss logic that determines whether requested data is found in the cache memory. L0 cache system 120 is the fastest memory device coupled to execution unit 118. In one embodiment, L0 cache system 120 is located on the same semiconductor die as execution unit 118, and data can be retrieved, for example, in approximately 4 clock cycles.

If data requested by execution unit 118 is not found in L0 cache system 120, execution unit 118 will attempt to retrieve the data from additional levels of memory devices through a memory request controller 119. After the L0 cache system 120, the next level of memory devices is an L1 cache system 122. Accessing L1 cache system 122 is typically 4-16 times as slow as accessing L0 cache system 120. In one embodiment, L1 cache system 122 is located on the same processor chip as execution unit 118, and data can be retrieved in approximately 24 clock cycles, for example.

If the data is not found in L1 cache system 122, execution unit 118 is forced to retrieve the data from the next level memory device, which is an external memory device coupled to an external bus 102. An external bus interface 124 is coupled to memory request controller 119 and external bus 102. The next level of memory device after L1 cache system 122 is an L2 cache system 106. Access to L2 cache system 106 is typically 4-16 times as slow as access to L1 cache system 122. In one embodiment, data can be retrieved from L2 cache system 106 in approximately 200 clock cycles.

After L2 cache system 106, the next level of memory device is main memory 104, which typically comprises dynamic random access memory ("DRAM"), and then disk memory 105. Access to main memory 104 and disk memory 105 is substantially slower than access to L2 cache system 106. In one embodiment, the computer system includes one external bus dedicated to L2 cache system 106, and another external bus used by all other external memory devices. In other embodiments of the present invention, processor 100 can include greater or less levels of memory devices than shown in Fig. 1. Disk memory 105, main memory 104 and L2 cache system 106 may be considered external memory because they are coupled to the processor 100 via external bus 102.

When attempting to load data to a register from memory, execution unit 118 first attempts to load the data from the first and fastest level of memory devices (i.e., L0 cache system 120), and then attempts to load the data from the second fastest level of memory (i.e., L1 cache system 122) and so on. Of course, the memory load takes an increasingly longer time as an additional memory
5 level is required to be accessed. When the data is finally found, the data retrieved by execution unit 118 is also stored in the lower levels of memory devices for future use.

For example, assume that a memory load instruction requires "data-1" to be loaded into a register. Execution unit 118 will first attempt to retrieve data-1 from L0 cache system 120. If it is not found there, execution unit 118 will next attempt to retrieve data-1 from L1 cache system 122. If it is not found there, execution unit 118 will next attempt to retrieve data-1 from L2 cache system 106. If data-1 is retrieved from L2 cache system 106, data-1 will then be stored in L1 cache system 122 and L0 cache system 120 in addition to being retrieved by execution unit 118.

A. General Description Of Replay System

Processor 100 further includes a replay system 117. Replay system 117 replays instructions
15 that were not executed properly when they were initially dispatched by scheduler 114. Replay system 117, like execution unit 118, receives instructions output by dispatch multiplexer 116. Execution unit 118 receives instructions from mux 116 over line 137, while replay system 117 receives instructions over line 139.

Replay system 117 includes two staging sections. One staging section a plurality of staging
20 queues A, B, C and D, while a second staging section is provided as staging queues E and F. Staging queues delay instructions for a fixed number of clock cycles. In one embodiment, staging queues

A-F each comprise one or more latches. The number of stages can vary based on the amount of staging or delay desired in each execution channel. Therefore, a copy of each dispatched instruction is staged through staging queues A-D in parallel to being staged through execution unit 118. In this manner, a copy of the instruction is maintained in the staging queues A-D and is provided to a checker 150, described below. This copy of the instruction may then be routed back to mux 116 for re-execution or “replay” if the instruction did not execute properly.

Replay system 117 further includes a checker 150 and a replay queue 170. Generally, checker 150 receives instructions output from staging queue D and then determines which instructions have executed properly and which have not. If the instruction has executed properly, the checker 150 declares the instruction “replay safe” and the instruction is forwarded to retirement unit 152 where instructions are retired in program order. Retiring instructions is beneficial to processor 100 because it frees up processor resources, thus allowing additional instructions to begin execution.

An instruction may execute improperly for many reasons. The most common reasons are a source dependency and an external replay condition. A source dependency can occur when a source of a current instruction is dependent on the result of another instruction. This data dependency can cause the current instruction to execute improperly if the correct data for the source is not available at execution time (i.e., the result of the other instruction is not available as source data at execution time).

A scoreboard 140 is coupled to the checker 150. Scoreboard 140 tracks the readiness of sources. Scoreboard 140 keeps track of whether the source data was valid or correct prior to

instruction execution. After the instruction has been executed, checker 150 can read or query the scoreboard 140 to determine whether data sources were not correct. If the sources were not correct at execution time, this indicates that the instruction did not execute properly (due to a data dependency), and the instruction should therefore be replayed.

5 Examples of an external replay condition may include a cache miss (e.g., source data was not found in L0 cache system 120 at execution time), incorrect forwarding of data (e.g., from a store buffer to a load), hidden memory dependencies, a write back conflict, an unknown data/address, and serializing instructions. The L0 cache system 120 generates a L0 cache miss signal 128 to checker 150 if there was a cache miss to L0 cache system 120 (which indicates that the source data for the instruction was not found in L0 cache system 120). Other signals can similarly be generated to checker 150 to indicate the occurrence of other external replay conditions. In this manner, checker 150 can determine whether each instruction has executed properly.

15 If the checker 150 determines that the instruction has not executed properly, the instruction will then be returned to multiplexer 116 to be replayed (i.e., re-executed). Each instruction to be replayed will be returned to mux 116 via one of two paths. Specifically, if the checker 150 determines that the instruction should be replayed, the Replay Queue Loading Controller 154 determines whether the instruction should be sent through a replay loop 156 including staging queues E and F, or whether the instruction should be temporarily stored in a replay queue 170 before returning to mux 116. Instructions routed via the replay loop 156 are coupled to mux 116 via line 20 161. Instructions can also be routed by controller 154 for temporary storage in replay queue 170 (prior to replay). The instructions stored in replay queue 170 are output or unloaded under control

of replay queue unloading controller 179. The instructions output from replay queue 170 are coupled to mux 116 via line 171. The operation of replay queue 170, Replay Queue Loading Controller 154 and Replay Queue Unloading Controller 179 are described in detail below.

In conjunction with sending a replayed instruction to mux 116, checker 150 sends a “stop scheduler” signal 151 to scheduler 114. According to an embodiment, stop scheduler signal 151 is sent to scheduler 114 in advance of the replayed instruction reaching the mux 116 (either from replay loop 156 or replay queue 170). In one embodiment, stop scheduler signal 151 instructs the scheduler 114 not to schedule an instruction on the next clock cycle. This creates an open slot or “hole” in the instruction stream output from mux 116 in which a replayed instruction can be inserted. A stop scheduler signal may also be issued from the replay queue unloading controller 179 to scheduler 114.

III. The Need For A Replay Queue

According to one embodiment, all instructions that did not execute properly (i.e., where checker 150 determined that the instructions were not replay safe) can be routed by controller 154 to mux 116 via replay loop 156 (including staging queues E and F). In such a case, all instructions, regardless of the type of instruction or the specific circumstances under which they failed to execute properly, will be routed back to the mux 116 via line 161 for replay. This works fine for short latency instructions which will typically require only one or a small number of passes or iterations through replay loop 156.

As noted above, the instructions of processor 100 may be speculatively scheduled for execution (i.e., before actually waiting for the correct source data to be available) on the expectation that the source data will be available for the majority of the memory load instructions (for example).

If it turns out that the source data was not available in L0 cache system 120 at the time of execution, (indicated by L0 cache miss signal 128 being asserted), the checker 150 determines that the instruction is not replay safe and sends the instruction back to mux 116 for replay.

During the period of time while the memory load instruction is being staged in staging queues E, F and A-D for replay, the execution unit 118 will attempt to retrieve the data from additional levels of memory devices through a memory request controller 119, and then store the retrieved data in L0 cache system 120 for the next iteration (the next execution attempt). A L0 cache miss, L1 cache hit may be considered to be a relatively common case for some systems.

According to an embodiment, the delay provided through the replay loop 156 (including through staging queues E-F and A-D) is designed or optimized for an L0 cache miss and a L1 cache hit. In other words, the delay provided through replay loop 156 is usually sufficient to allow data to be retrieved from the L1 cache system and stored back in the L0 cache system 120 prior to execution the second time (i.e., assuming a L0 cache miss and a L1 cache hit on the first execution of the instruction). For relatively short latency instructions like these (e.g., where there was a L0 cache miss and a L1 cache hit), only one or few iterations through the replay loop 156 will typically be required before the instruction will execute properly.

However, there may be one or more long latency instructions which will require many iterations through the replay loop 156 before finally executing properly. If the instruction did not execute properly on the first attempt, the checker 150 may determine whether the instruction requires a relatively long period of time to execute (i.e., a long latency instruction), requiring several passes

through the replay loop 156 before executing properly. There are many examples of long latency instructions. One example is a divide instruction which may require many clock cycles to execute.

Another example of a long latency instruction is a memory load or store instruction where there was an L0 cache system miss and an L1 cache system miss. In such a case, an external bus request will be required to retrieve the data for the instruction. If access across an external bus is required to retrieve the desired data, the access delay is substantially increased. To retrieve data from an external memory, the memory request controller 119 may be required to arbitrate for ownership of the external bus 102 and then issue a bus transaction (memory read) to bus 102, and then await return of the data from one of the external memory devices. As an example, according to an embodiment, approximately 200 clock cycles may be required to retrieve data from a memory device on an external bus versus 4-24 clock cycles to retrieve data from L0 cache system 120 or L1 cache system 122. Thus, due to the need to retrieve data from an external memory device across the external bus 102, this load instruction where there was a L1 cache miss may be considered to be a long latency instruction.

During this relatively long period of time while the long latency instruction is being processed (e.g., while the data is being retrieved across the external bus 102 for a L1 cache miss), the instruction may circulate tens or even hundreds of iterations through the replay loop 156. Each time the long latency instruction is replayed before the source data has returned, this instruction unnecessarily occupies a slot in the output of mux 116 and uses execution resources which could have been allocated to other instructions which are ready to execute properly. Moreover, there may be many additional instructions which are dependent upon the result of this long latency load

instruction. As a result, each of these dependent instructions also will similarly repeatedly circulate through the replay loop 156 without properly executing. All of these dependent instructions will not execute properly until after the data for the long latency instruction returns from the external memory device, occupying and wasting even additional execution resources. Thus, the many unnecessary and excessive iterations through the replay loop 156 before the return of the data wastes valuable resources, wastes power and increases the application latency.

For example, where several calculations are being performed for displaying pixels on a display, an instruction for one of the pixels may be a long latency instruction, e.g., requiring a memory access to an external memory device. There may be many non-dependent instructions for other pixels behind this long latency instruction that do not require an external memory access. As a result, by continuously replaying the long latency instruction and its many dependent instructions thereon, the non-dependent instructions for the other pixels may be precluded from execution. Once the long latency instruction has properly executed, execution slots and resources become available and the instructions for the other pixels can then be executed. An improved solution would be to allow the non-dependent instructions to execute in parallel while the long latency instruction awaits return of its data.

According to an embodiment, an advantageous solution to this problem is to temporarily store the long latency instruction in a replay queue 170 along with its dependent instructions. When the data for the long latency instruction returns from the external memory device, the long latency instruction and its dependent instructions can then be unloaded from the replay queue 170 and sent

to mux 116 for replay. In this manner, the long latency instruction will typically not “clog” or unnecessarily delay execution of other non-dependent instructions.

Therefore, the advantages of using a replay queue in this manner include:

a) prudent and efficient use of execution resources - execution resources are not wasted on instructions which have no hope of executing properly at that time;

b) power savings - since power is not wasted on executing long latency instructions before their data is available;

c) reduce overall latency of application - since independent instructions are permitted to execute in parallel while the data is being retrieved from external memory for the long latency instruction; and

d) instructions having different and unknown latencies can be accommodated using the same hardware because, according to an embodiment, the instruction in the replay queue will be executed upon return of the data (whenever that occurs).

IV. Operation Of the Replay Queue and Corresponding Control Logic

According to an embodiment, a long latency instruction is identified and loaded into replay queue 170. One or more additional instructions (e.g., which may be dependent upon the long latency instruction) may also be loaded into the replay queue 170. When the condition causing the instruction to not complete successfully is cleared (e.g., when the data returns from the external bus after a cache miss or after completion of a division or multiplication operation or completion of another long latency instruction), the replay queue 170 is then unloaded so that the long latency instruction and the others stored in replay queue 170 may then be re-executed (replayed).

According to one particular embodiment, replay queue loading controller 154 detects a L1 cache miss (indicating that there was both a L0 cache miss and a L1 cache miss). As shown in the example embodiment of Fig. 1, L1 cache system 122 detects a L1 cache miss and generates or outputs a L1 cache miss signal 130 to controller 154. Because there was also a L0 cache miss, L0 cache miss signal 128 is also asserted (an external replay condition), indicating to checker 150 that the instruction did not execute properly. Because the instruction did not execute properly, checker 150 provides the instruction received from staging queue D to replay queue loading controller 154. Controller 154 must then determine whether to route the replay instruction to mux 116 via replay loop 156 or via replay queue 170.

According to an embodiment, if the replay queue loading controller 154 determines that the instruction is not a long latency instruction, the instruction is sent to mux 116 for replay via replay loop 156. However, if controller 154 determines that the instruction is a long latency instruction (e.g., where an external memory access is required), the controller 154 will load the instruction into replay queue 170. In addition, replay queue loading controller 154 must also determine what instructions behind the long latency (or agent) instruction should also be placed into replay queue 170. Preferably, all instructions that are dependent upon the long latency instruction (or agent instruction) should also be placed in the replay queue 170 because these will also not execute properly until return of the data for the agent instruction. However, it can sometimes be difficult to identify dependent instructions because there can be hidden memory dependencies, etc. Therefore, according to an embodiment, once the long latency or agent instruction has been identified and loaded into the replay queue 170, all additional instructions which do not execute properly and have

a sequence number greater than that of the agent instruction (i.e., are programmatically younger than the agent instruction) will be loaded into the replay queue 170 as well.

Replay queue unloading controller 179 preferably receives a signal when the condition causing the instruction to not complete or execute successfully has been cleared (e.g., when the long latency instruction in the replay queue 170 is ready to be executed). As an example, when the data for the long latency instruction returns from the external memory device, the external bus interface 124 asserts the data return signal 126 to replay queue unloading controller 179. Replay queue unloading controller 179 then unloads the instruction(s) stored in the replay queue 170, e.g., in a first-in, first-out (FIFO) manner, to mux 116 for replay (re-execution). The expectation is that the long latency instruction (and its dependents) will now properly execute because the long latency instruction is ready to be executed (e.g., the source data for the long latency instruction is now available in L0 cache system 120).

A. Arbitration/Priority

As described above, mux 116 will receive instructions from three sources: instructions from scheduler 114, instructions provided via line 161 from replay loop 156 and instructions provided via line 171 which are output from replay queue 170 (e.g., after return of the source data for the agent instruction). However, mux 116 can output or dispatch only one instruction per execution port at a time to execution unit 118. Therefore, an arbitration (or selection) mechanism should be provided to determine which of three instruction paths should be output or selected by mux 116 in the event instructions are provided on more than one path. If instructions are provided only from scheduler 114, then the instructions provided over line 115 from scheduler 114 are the default selection for

mux 116.

According to an embodiment, the checker 150, controller 154 and controller 179 can arbitrate to decide which path will be selected for output by mux 116. Once the checker 150 and controllers 154 and 175 have determined which path will be selected for output, the replay loop select signal 163 may be asserted to select the instruction from the replay loop 156, or the replay queue select signal 175 may be asserted to select the instruction output from the replay queue 170. If the instruction path from scheduler 114 is selected for output, then neither select signal 163 nor select signal 179 will be asserted (indicating the default selection from scheduler 114).

Checker 150 and controllers 154 and 179 may use any of several arbitration algorithms to determine which of three instruction paths should be output or selected by mux 116 in the event instructions are provided on more than one path. A couple of example arbitration (or selection) algorithms will be described, but the present invention is not limited thereto.

1. Fixed Priority Scheme

According to one embodiment, a fixed priority scheme may be used, for example, where the replay queue 170 is given priority over the replay loop 156, which is given priority over the scheduler 114. Other fixed priority schemes may be used as well.

2. Age Priority Scheme

A second possible arbitration algorithm is where the oldest instruction is given priority for execution (i.e., oldest instruction is selected by mux 116) regardless of the path. In this embodiment, checker 150 and controllers 154 and 179 may compare the age of an instruction in the replay loop 156 to the age of an instruction to be output from the scheduler 114 to the age of an instruction to

be output from the replay queue 170 (assuming an instruction is prepared to be output from the replay queue 170). According to an embodiment, the age comparison between instructions may be performed by comparing sequence numbers of instructions, with a smaller or lower sequence number indicating a programmatically older (or preceding) instruction, which would be given priority in this scheme. In the event that an instruction is output from checker 150 to be replayed and an instruction is output from replay queue 170 to mux 116 for execution, the replayed instruction output from checker 150 may be stored in the replay queue 170.

B. Example Instruction Format

Fig. 3 is a diagram illustrating an example format of an instruction provided in a replay path according to an embodiment. As shown in Fig. 3, the instruction that is staged along the replay path (e.g., beginning at line 137) may include several fields, such as the sources (source1 302 and source2 304), a destination 306 and an operation field that identifies the operation to be performed (e.g., memory load). A sequence number 310 is also provided to identify the age or order of the instructions.

C. Another Example

Fig. 2 is a flow chart illustrating an example operation of instruction processing. At block 205, an instruction is output by mux 116 (from one of the three paths). At block 210, the instruction is executed by execution unit 118. At block 215, checker 150 determines whether the instruction executed properly or not. If the instruction executed properly (i.e., the instruction is “replay safe”), the instruction is sent to retirement unit 152, block 220. If the instruction did not execute properly (e.g., failed replay), then the process proceeds to block 225.

At block 225, it is determined whether the instruction is an agent instruction (or a long latency instruction). One example way that this is performed is by replay queue loading controller 154 receiving a L1 cache miss signal 130 if there is a L1 cache miss. There are other instances where a long latency or agent instruction can be detected (such as a divide instruction). If this instruction is an agent or long latency instruction, the instruction is loaded into replay queue 170, block 245.

If the instruction is not an agent instruction, process proceeds to block 230. At block 230, the controller 154 determines if there is already an agent instruction in the replay queue. If there is no agent instruction in queue 170, the instruction is placed into the replay loop 156 for replay, block 250.

Next, the checker 150 and/or controller 154 determines whether this instruction is younger than the agent instruction in the replay queue, by comparing sequence numbers of the two instructions. If the instruction is younger than the agent instruction in the replay queue 170, the instruction is then loaded into the replay queue 170 to wait until the agent instruction is ready to be properly executed or when the condition that caused the agent to improperly execute to be cleared or resolved (e.g., to wait until the data for the agent returns from the external memory device).

It is also possible for multiple agent instructions to be loaded into replay queue. In such case, each agent instruction and its dependent instructions in the queue may be unloaded based on the agent being able to execute properly (e.g., source data for the agent returning from an external memory device). According to one embodiment, all instructions in the replay queue 170 may be unloaded when first agent instruction in the queue 170 is ready to be executed properly (e.g., when

the data has returned from the external bus). In an alternative embodiment, only those dependent instructions stored in the replay queue 170 after the agent that is ready to execute and before the next agent are unloaded when the agent is ready to execute properly. In the case of multiple agent instructions, the steps of Fig. 2 may be performed in parallel for each agent instruction.

5 Therefore, it can be seen from the embodiment of Fig. 2, that a (non-agent) instruction is placed in the replay queue 170 if three conditions are met (according to an example embodiment):

a) the instruction did not properly execute (otherwise, the instruction will be retired, not replayed); and

b) there is already an agent instruction in the replay queue 170 (an active agent);and

10 c) the instruction is programmatically younger than the agent instruction in the replay queue 170 (i.e., a greater sequence number than the agent).

Several embodiments of the present invention are specifically illustrated and/or described herein. However, it will be appreciated that modifications and variations of the present invention are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.

15

WHAT IS CLAIMED IS:

1 1. A processor comprising:
2 an execution unit to execute instructions;
3 a replay system coupled to the execution unit to replay instructions which have not executed
4 properly, the replay system comprising:
5 a checker to determine whether each instruction has executed properly; and
6 a replay queue coupled to the checker to temporarily store one or more instructions
7 for replay.

1 2. The processor of claim 1 wherein said replay system further comprises:
2 a replay loop to route an instruction which executed improperly to an execution unit for
3 replay; and
4 a replay queue loading controller to determine whether to load an improperly executed
5 instruction to the replay loop or into the replay queue.

1 3. The processor of claim 2 and further comprising:
2 a scheduler to output instructions; and
3 a multiplexer or selection mechanism having a first input coupled to the scheduler, a second
4 input coupled to the replay loop and a third input coupled to an output of the replay queue.

1 4. The processor of claim 1 wherein said replay queue comprises a replay queue coupled to
2 the checker to temporarily store one or more long latency instructions until the instruction is ready
3 for execution.

5. The processor of claim 1 wherein the replay queue comprises a replay queue coupled to
the checker to temporarily store an instruction that is not ready to execute properly, the instruction
being unloaded from the replay queue when the instruction is ready to execute properly.

6. The processor of claim 1 wherein the replay queue comprises a replay queue coupled to
the checker to temporarily store an instruction in which source data must be retrieved from an
external memory device, the instruction being unloaded from the replay queue when the source data
for the instruction returns from the external memory device.

7. The processor of claim 1 wherein said execution unit is a memory load unit, the processor
further comprising:

a first level cache system coupled to the memory load unit;
a second level cache system coupled to the first level cache system; and
wherein the memory load unit performs a data request to external memory if there is a miss
on both the first level and second level cache systems.

1 8. The processor of claim 7 wherein a load instruction will be loaded into the replay queue
2 when there is a miss on both the first level and second level cache systems, and the load instruction
3 is unloaded from the replay queue for re-execution when the data for the instruction returns from the
4 external memory.

1 9. A processor comprising:
2 a multiplexer having an output;
3 a scheduler coupled to a first input of the multiplexer;
4 an execution unit coupled to an output of the multiplexer;
5 a checker coupled to the output of the multiplexer to determine whether an instruction has
6 executed properly;
7 a replay queue to temporarily store instructions, an output of the replay queue coupled to a
8 second input of the multiplexer; and
9 a controller coupled to the checker to determine when to load an instruction into the replay
10 queue and to determine when to unload the replay queue.

1 10. The processor of claim 9 and further comprising a staging section coupled between the
2 checker and a third input to the multiplexer to provide a replay loop, the controller controlling the
3 multiplexer to select either the output of the scheduler, the replay loop or the output of the replay
4 queue.

11. The processor of claim 9 wherein the controller loads an instruction into the replay queue when the instruction is not ready to execute properly, and unloads the instruction from the replay queue when the instruction is ready to execute properly.

1 12. The processor of claim 9 wherein the controller determines when to unload the replay
2 queue based on a data return signal.

1 13. A method of processing instructions comprising:
2 dispatching an instruction where the instruction to an execution unit and to a replay system;
3 determining whether the instruction executed properly;
4 if the instruction did not execute properly, then:
5 determining whether the instruction should be routed back for re-execution or whether
6 the instruction should be temporarily stored in a queue.

1 14. A method of processing instructions comprising:
2 dispatching an instruction where the instruction is received by an execution unit and a replay
3 system;
4 determining whether the instruction executed properly;
5 if the instruction did not execute properly, then:
6 routing the instruction to the execution unit for re-execution if the instruction is a first
7 type of instruction;

8 otherwise, loading the instruction into a replay queue if the instruction is a second
9 type of instruction.

1 15. The method of claim 14 wherein the first type of instruction comprises a short latency
2 instruction, and the second type of instruction is a longer latency instruction.

1 16. A method of processing instructions comprising:
2 dispatching an instruction to an execution unit;
3 determining whether the instruction executed properly;
4 if the instruction did not execute properly, then:
5 determining whether an access across an external bus is required for proper
6 instruction execution;
7 routing the instruction to the execution unit for re-execution if an external bus access
8 is not required;
9 otherwise temporarily storing the instruction in a replay queue if an external bus
10 access is required, then unloading the instruction from the replay queue to the execution unit for
11 execution when the access across an external bus has been completed.

1 17. A method of processing instructions comprising:
2 dispatching a load instruction to a memory load unit;
3 determining whether the load instruction executed properly;

4 if the load instruction did not execute properly, then:
5 determining whether there was a cache miss to all on-chip cache systems;
6 routing the load instruction to the memory load unit for re-execution if there was not
7 a miss to all on-chip cache systems;
8 otherwise, if there was a cache miss to all on-chip cache systems, then temporarily
9 storing the load instruction in a replay queue and retrieving the data for the load instruction from
10 external memory, and then unloading the load instruction from the replay queue to the memory load
11 unit for re-execution when the data for the load instruction has been retrieved from the external
12 memory.

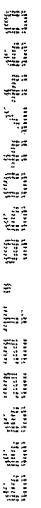
1 18. A method of processing instructions comprising:
2 receiving an improperly executed instruction;
3 determining whether the instruction is an agent instruction;
4 if the instruction is an agent instruction, then loading the instruction into a replay queue;
5 otherwise, if the instruction is not an agent instruction, then loading the instruction into the
6 replay queue if the following conditions are met:
7 a) there is already an agent instruction in the replay queue; and
8 b) the instruction is younger than the agent instruction in the replay queue; and
9 unloading one or more of the instructions in the replay queue to an execution unit when the
10 agent instruction in the replay queue is ready to execute.

- 1 19. The method of claim 18 wherein said step of unloading comprises the step of unloading
- 2 the instructions in the replay queue to an execution unit when data for the agent instruction is
- 3 available to allow the instruction to execute properly.

Abstract

A processor is provided that includes an execution unit for executing instructions and a replay system for replaying instructions which have not executed properly. The replay system is coupled to the execution unit and includes a checker for determining whether each instruction has
5 executed properly and a replay queue coupled to the checker for temporarily storing one or more instructions for replay. The replay queue may be used to store a long latency instruction, such as a load in which data must be retrieved from an external memory device. The long latency instruction and possibly one or more dependent instruction are stored in the replay queue until the long latency instruction is ready to be executed (e.g., data for the load instruction has been retrieved from external memory). Once the long latency instruction is ready to be executed, (e.g., the data is available), the long latency instruction may then be unloaded from the replay queue for re-execution.

The diagram illustrates a memory hierarchy. At the top, a box labeled "Disk Memory 105" is connected to a thick horizontal line representing the "External Bus 102". Below it, a box labeled "Main Memory 104" is connected to the bus via a diamond-shaped symbol. To the right of the Main Memory box, a bracket groups it with the L2 Cache box as "External memory". Below the Main Memory box, a box labeled "L2 Cache Sys. 106" is connected to the Main Memory box via a double-headed arrow. The L2 Cache box is also connected to the External Bus 102 via a diamond-shaped symbol.



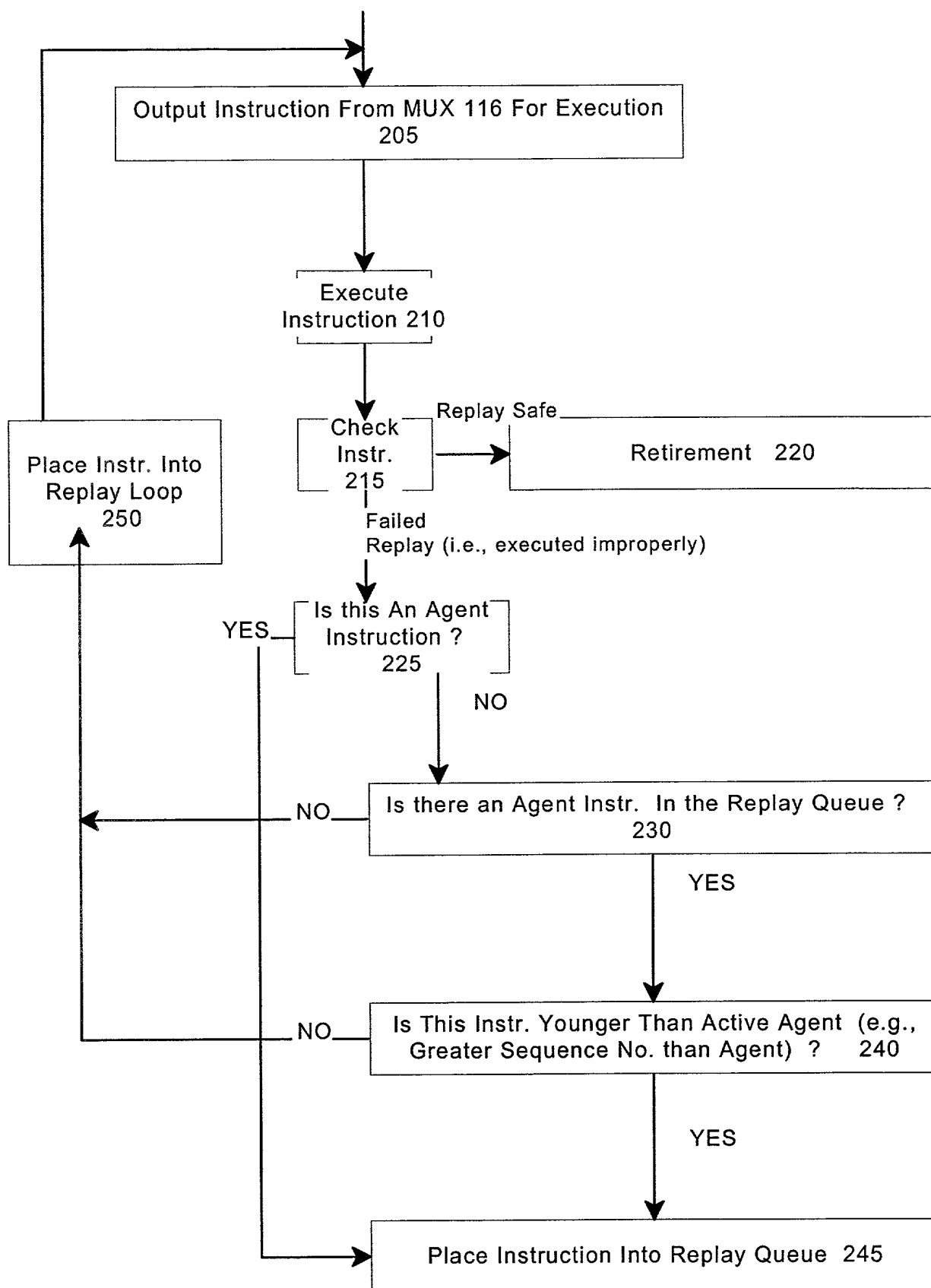


FIG. 2

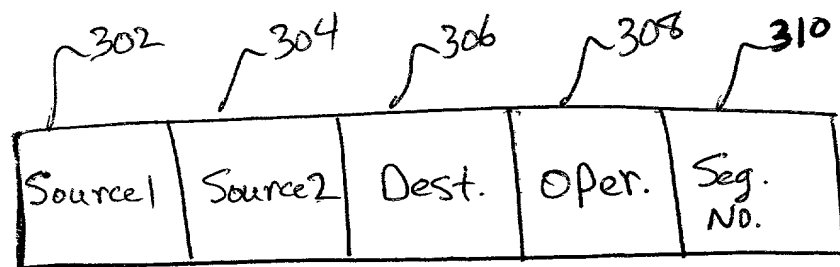


Fig. 3

Attorney's Docket No.: 219.37081X00

PATENT

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION
(FOR INTEL CORPORATION PATENT APPLICATIONS)

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below, next to my name.

I believe I am the original, first, and sole inventor (if only one name is listed below) or an original, first, and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

**A PROCESSOR WITH A REPLAY SYSTEM THAT INCLUDES
A REPLAY QUEUE FOR IMPROVED THROUGHPUT**

the specification of which

 X is attached hereto.

 was filed on as

United States Application Number _____

or PCT International Application Number _____

and was amended on _____
(if applicable)

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claim(s), as amended by any amendment referred to above. I do not know and do not believe that the claimed invention was ever known or used in the United States of America before my invention thereof, or patented or described in any printed publication in any country before my invention thereof or more than one year prior to this application, that the same was not in public use or on sale in the United States of America more than one year prior to this application, and that the invention has not been patented or made the subject of an inventor's certificate issued before the date of this application in any country foreign to the United States of America on an application filed by me or my legal representatives or assigns more than twelve months (for a utility patent application) or six months (for a design patent application) prior to this application.

I acknowledge the duty to disclose all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119(a)-(d), of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)

Priority
Claimed

<u> </u> (Number)	<u> </u> (Country)	<u> </u> (Day/Month/Year Filed)	<u> </u> Yes	<u> </u> No
---	--	---	----------------------	---------------------

<u> </u> (Number)	<u> </u> (Country)	<u> </u> (Day/Month/Year Filed)	<u> </u> Yes	<u> </u> No
---	--	---	----------------------	---------------------

<u> </u> (Number)	<u> </u> (Country)	<u> </u> (Day/Month/Year Filed)	<u> </u> Yes	<u> </u> No
---	--	---	----------------------	---------------------

I hereby claim the benefit under title 35, United States Code, Section 119(e) of any United States provisional application(s) listed below

<u> </u> (Application Number)	<u> </u> Filing Date
---	--

<u> </u> (Application Number)	<u> </u> Filing Date
---	--

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application:

<u>08/746,547</u> (Application Number)	<u>November 13, 1996</u> Filing Date	<u>Patented</u> (Status -- patented, pending, abandoned)
---	---	---

<u>09/106,857</u> (Application Number)	<u>June 30, 1998</u> Filing Date	<u>Pending</u> (Status -- patented, pending, abandoned)
---	-------------------------------------	--

I hereby appoint: Donald R. Antonelli, Reg. No. 20,296; David T. Terry, Reg. No. 20,178; Melvin Kraus, Reg. No. 22,466; William I. Solomon, Reg. No. 28,565; Gregory E. Montone, Reg. No. 28,141; Ronald J. Shore, Reg. No. 28,577; Donald E. Stout, Reg. No. 26,422; Alan E. Schiavelli, Reg. No. 32,087; James N. Dresser, Reg. No. 22,973; Carl I. Brundidge, Reg. No. 29,621; Paul J. Skwierawski, Reg. No. 32,173, my attorneys; of ANTONELLI, TERRY, STOUT & KRAUS, LLP with offices located at 1300 North Seventeenth Street, Suite 1800, Arlington, Virginia 22209, telephone: (703) 312-6600, fax: (703) 312-6666; and Alan K. Aldous, Reg. No. 31,905; Robert D. Anderson, Reg. No. 33,826; Joseph R. Bond, Reg. No. 36,458; Richard C. Calderwood, Reg. No. 35,468; Jeffrey S. Draeger, Reg. No. 41,000; Cynthia Thomas Faatz, Reg. No. 39,973; Sean Fitzgerald, Reg. No. 32,027; Seth Z. Kalson, Reg. No. 40,670; David J. Kaplan, Reg. No. 41,105; Leo V. Novakoski, Reg. No. 37,198; Naomi Obinata, Reg. No. 39,320; Thomas C. Reynolds, Reg. No. 32,488; Steven P. Skabrat, Reg. No. 36,279; Howard A. Skaist, Reg. No. 36,008; Steven C. Stewart, Reg. No. 33,555; Raymond J. Werner, Reg. No. 34,752; and Charles K. Young, Reg. No. 39,435; my patent attorneys, and Calvin E. Wells, Reg. No. P43,256; and Alexander Ulysses Witkowski, Reg. No. P43,280; my patent agents, of INTEL CORPORATION; with full power of substitution and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected herewith.

Send all correspondence to:

ANTONELLI, TERRY, STOUT & KRAUS, LLP
1300 North Seventeenth Street
Suite 1800
Arlington, VA. 22209

Direct all telephone calls and faxes to:

TEL: (703) 312-6600
FAX: (703) 312-6666

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Sole/First Inventor Amit A. Merchant

Inventor's Signature *Amit Merchant* Date 12/8/99
Residence Portland, Oregon Citizenship U.S.A.
(City, State) (Country)
Post Office Address 5468 N.W. Deerfield Way, Portland Oregon 97229

Full Name of Second/Joint Inventor Darrell D. Boggs

Inventor's Signature *Darrell D. Boggs* Date 12/9/99
Residence Aloha, Oregon Citizenship U.S.A.
(City, State) (Country)
Post Office Address 2200 S.W. 195th, Aloha, Oregon 97006

Post Office Address 9540 N.W. Skyview Drive, Portland, Oregon 97231

INTEL CORPORATION

Title 37, Code of Federal Regulations, Section 1.56
Duty to Disclose Information Material to Patentability

(a) A patent by its very nature is affected with a public interest. The public interest is best served, and the most effective patent examination occurs when, at the time an application is being examined, the Office is aware of and evaluates the teachings of all information material to patentability. Each individual associated with the filing and prosecution of a patent application has a duty of candor and good faith in dealing with the Office, which includes a duty to disclose to the Office all information known to that individual to be material to patentability as defined in this section. The duty to disclose information exists with respect to each pending claim until the claim is cancelled or withdrawn from consideration, or the application becomes abandoned. Information material to the patentability of a claim that is cancelled or withdrawn from consideration need not be submitted if the information is not material to the patentability of any claim remaining under consideration in the application. There is no duty to submit information which is not material to the patentability of any existing claim. The duty to disclose all information known to be material to patentability is deemed to be satisfied if all information known to be material to patentability of any claim issued in a patent was cited by the Office or submitted to the Office in the manner prescribed by §§ 1.97(b)-(d) and 1.98. However, no patent will be granted on an application in connection with which fraud on the Office was practiced or attempted or the duty of disclosure was violated through bad faith or intentional misconduct. The Office encourages applicants to carefully examine:

- (1) Prior art cited in search reports of a foreign patent office in a counterpart application, and
 - (2) The closest information over which individuals associated with the filing or prosecution of a patent application believe any pending claim patentably defines, to make sure that any material information contained therein is disclosed to the Office.
- (b) Under this section, information is material to patentability when it is not cumulative to information already of record or being made of record in the application, and
- (1) It establishes, by itself or in combination with other information, a prima facie case of unpatentability of a claim; or
 - (2) It refutes, or is inconsistent with, a position the applicant takes in:
 - (i) Opposing an argument of unpatentability relied on by the Office, or
 - (ii) Asserting an argument of patentability.

A prima facie case of unpatentability is established when the information compels a conclusion that a claim is unpatentable under the preponderance of evidence, burden-of-proof standard, giving each term in the claim its broadest reasonable construction consistent with the specification, and before any consideration is given to evidence which may be submitted in an attempt to establish a contrary conclusion of patentability.

- (c) Individuals associated with the filing or prosecution of a patent application within the meaning of this section are:
- (1) Each inventor named in the application;
 - (2) Each attorney or agent who prepares or prosecutes the application; and
 - (3) Every other person who is substantively involved in the preparation or prosecution of the application and who is associated with the inventor, with the assignee or with anyone to whom there is an obligation to assign the application.
- (d) Individuals other than the attorney, agent or inventor may comply with this section by disclosing information to the attorney, agent, or inventor.